EECS442_HW1

Yuzhen Chen

January 2024

NumPy Intro

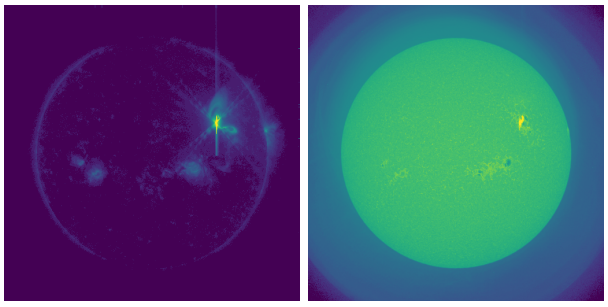
1.1 - Terminal Output

```
Running w1
Running w2
Running w3
Running w4
Running w5
Running w6
Running w7
Running w8
Running w9
Running w10
Running w11
Running w12
Running w13
Running w14
Running w15
Running w16
Running w17
Running w18
Running w19
Running w20
Ran warmup tests
20/20 = 100.0
```

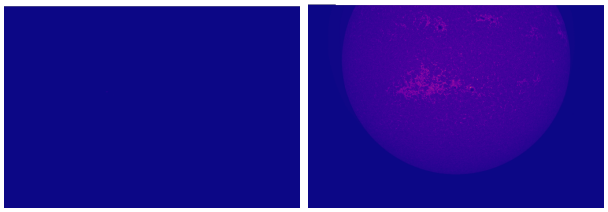
```
Running t1
Running t2
Running t3
Running t4
Running t5
Running t6
Running t7
Running t8
Running t9
Running t10
Running t11
Running t12
Running t13
Running t14
Running t15
Running t16
Running t17
Running t18
Running t19
Running t20
Ran all tests
20/20 = 100.0
```

Data Interpretation and Visualization

2.1 - 2 images from mysterydata2.npy



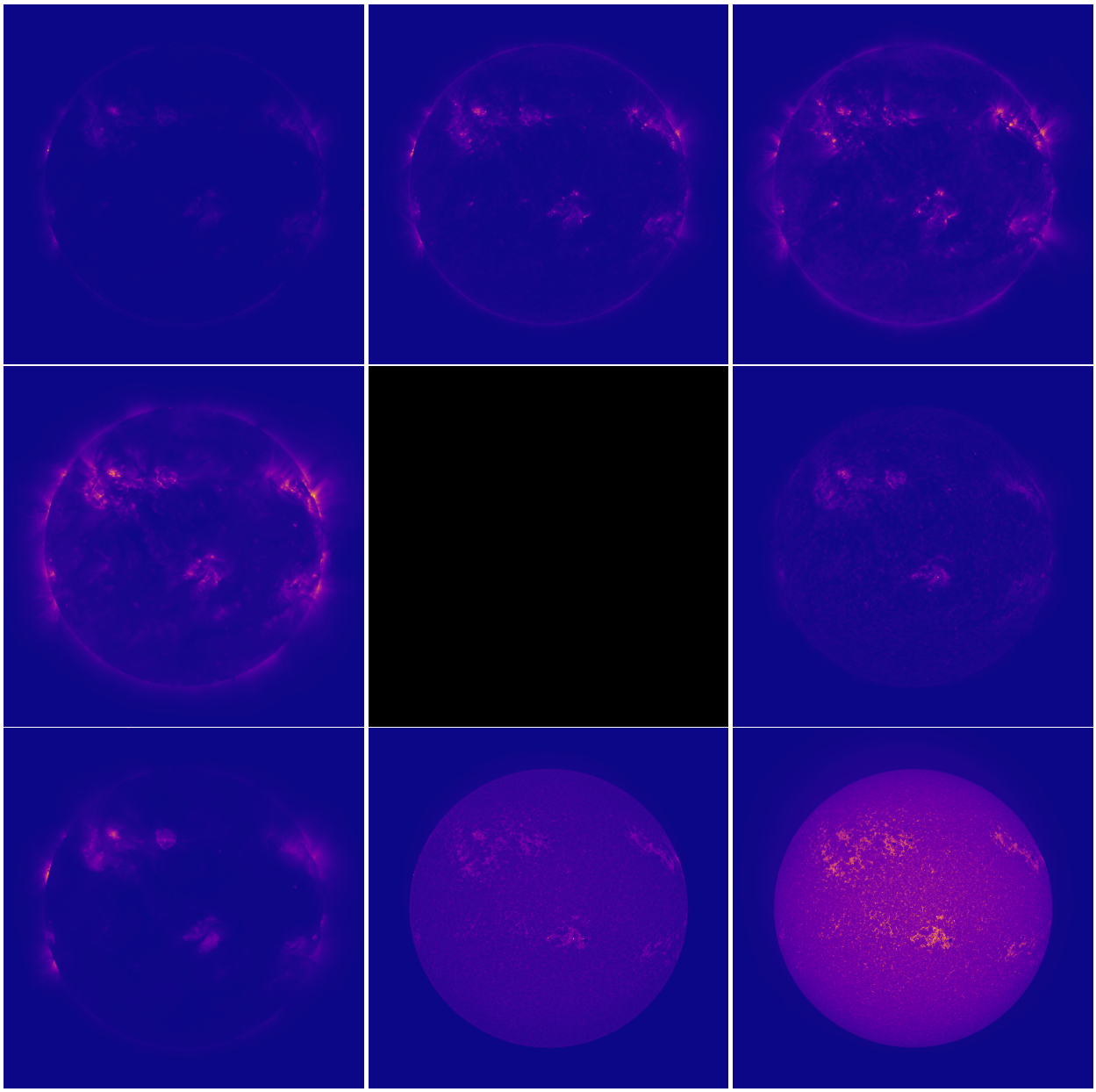
2.2 - 2 images from mysterydata3.npy



2.3 - colorMapArray

```
def colormapArray(X, colors):
    """
    Basically plt.imsave but return a matrix instead

    Given:
        a HxW matrix X
        a Nx3 color map of colors in [0,1] [R,G,B]
    Outputs:
        a HxW uint8 image using the given colormap.
           See the Bewares
    """

    v_min = np.nanmin(X)
    v_max = np.nanmax(X)

     # Check for divide-by-zero
    if v_max == v_min:
        # Handle the case where v_max and v_min are
           equal
        return np.zeros_like(X, dtype=np.uint8)


    N = colors.shape[0]
    R = (N-1)*(X-v_min)/(v_max-v_min)

    image = colors[R.astype(int)]

    image_uint8 = (image*255).astype(np.uint8)

    return image_uint8
```

2.4 - 9 images from mysterydata4.npy



Lights on a Budget

3.1 Naive Approach

1 - quantize

```
    def quantize(v, palette):
    """
    Given a scalar v and array of values palette,
    return the index of the closest value
    """
    if v.ndim == 0:
        min_in_palette = np.argmin(np.abs(palette-v))
```

2 - quantizeNaive

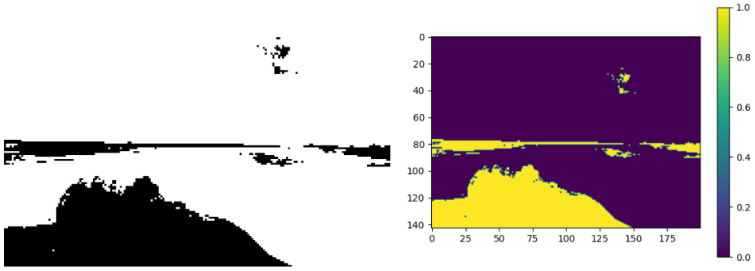
```
    def quantizeNaive(IF, palette):
    """Given a floating-point image return quantized
       version (Naive)"""
    # quantizing multiple
    output = np.zeros(IF.shape,dtype=np.uint8)
    row_num = IF.shape[0]
    col_num = IF.shape[1]

    for i in range(row_num):
        for j in range(col_num):
            output[i][j] = quantize(IF[i][j],palette)
    return output
```

3 - Quantize Runtime

Q: If you apply this to the folder gallery, why might your code (that calls quantize) take a very long time?

A: since the image in the folder gallery is larger, so the helper function quantize() will be used more time.

4 - Intensity Values vs Palette Values



Q: Do low intensity values correspond to low palette values?

A: Yes, the lower intensity value will correspond to the low palette index, and the low palette index will then correspond to the low pallete values.

5 - Two input/output pairs: aep.jpg + your choice

3.2 Floyd-Steinberg

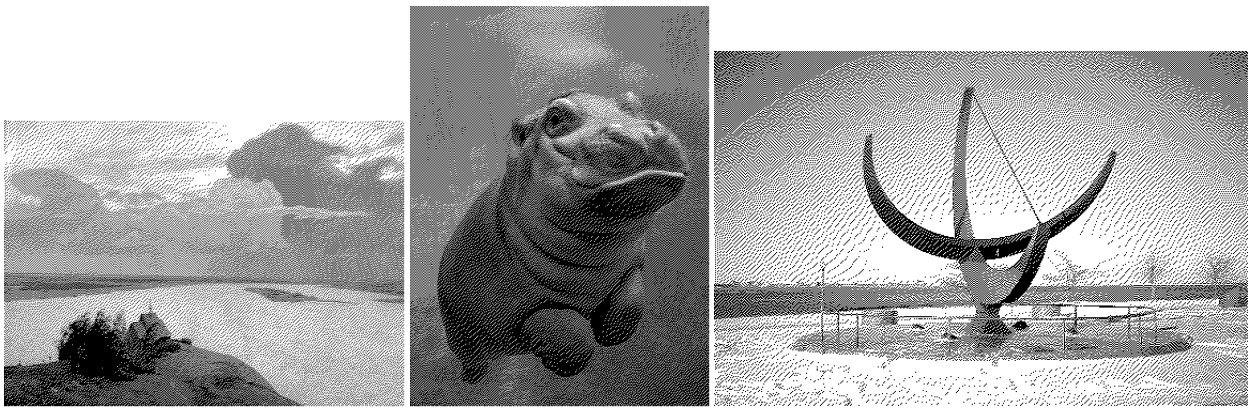
1 - quantizeFloyd

```
        def quantizeFloyd(IF, palette):
    """
    Given a floating-point image return quantized
       version (Floyd-Steinberg)
    """
    output = np.zeros(IF.shape,dtype=np.uint8)
    row_num = IF.shape[0]
    col_num = IF.shape[1]
    # channel_num = IF.shape[2] #----->for vector
    pixel = IF.copy()

    for i in range(row_num-1):
        for j in range(col_num-1):
            old_value = pixel[i][j]
            color_index = quantize(old_value,palette)
            # print("color_index",color_index)
            output[i][j] = color_index

            new_value = palette[color_index]
               #----->for scaler

            # # print("channel number is: ",
               channel_num) #------>for vector
            # new_value = np.zeros((channel_num))
            # for k in range(channel_num):
            #     # print("--->",color_index[1])
            #     temp_index = int(color_index[k])
            #     # print("temp_index", temp_index)
            #     new_value[k] = palette[temp_index]
            # # # exit(0)

            error = old_value-new_value

            pixel[i+1][j]   += error*7/16
            pixel[i-1][j+1] += error*3/16
            pixel[i][j+1]   += error*5/16
            pixel[i+1][j+1] += error*1/16


    return output
```

2 - Why does dithering work?

A: Introducing controlled noise or patterns into an image can help to simulate additional colors or shades, which helps mitigate the visual artifacts caused by reducing the number of available colors. At the same time, this technique exploits the human eye's ability to blend small, varied patterns into perceived smoother tones, enhancing the overall appearance of images, especially in situations with limited color representation.

3 - 3 results from gallery/ including aep.jpg



3.3 Resizing Images

1 - resizeToSquare

```
def resizeToSquare(I, maxDim):
"""Given an image, make sure it's no bigger than
   maxDim on either side"""

height, width = I.shape[:2]

max_size = max(height, width)

if max_size < maxDim:
    return I
else:
    ratio = maxDim / max_size
    if (max_size == height):

        new_height = maxDim
        new_width = ratio * width
    else:
        new_width = maxDim
```

```
            new_height = ratio * height

    old_image = I.copy()
    new_image = cv2.resize(old_image,(int(new_width),
        int(new_height)))

    return new_image
```

3.4 Handling Color

1 - quantize (scalar and vector)

```
    def quantize(v, palette):
    """
    Given a scalar v and array of values palette,
    return the index of the closest value
    """
    if v.ndim == 0:
        min_in_palette = np.argmin(np.abs(palette-v))
        return min_in_palette
    else:
        min_in_palette = np.zeros((v.shape[0]))
        # print("v_dim--->",v.ndim)
        for i in range(v.shape[0]):
            min_in_palette[i] =
                np.argmin(np.abs(palette-v[i]))
            # print("time: ",i)
        return min_in_palette
```

2 - quantizeFloyd (multi-channel)

```
    def quantizeFloyd(IF, palette):
    """
    Given a floating-point image return quantized
        version (Floyd-Steinberg)
    """
    output = np.zeros(IF.shape,dtype=np.uint8)
    row_num = IF.shape[0]
    col_num = IF.shape[1]
    channel_num = IF.shape[2] #----->for vector
    pixel = IF.copy()

    for i in range(row_num-1):
        for j in range(col_num-1):
```

```
            old_value = pixel[i][j]
            color_index = quantize(old_value,palette)
            # print("color_index",color_index)
            output[i][j] = color_index

            # new_value = palette[color_index]
               #----->for scaler

            # print("channel number is: ",
               channel_num) #------>for vector
            new_value = np.zeros((channel_num))
            for k in range(channel_num):
                # print("--->",color_index[1])
                temp_index = int(color_index[k])
                # print("temp_index", temp_index)
                new_value[k] = palette[temp_index]
            # # exit(0)

            error = old_value-new_value

            pixel[i+1][j]   += error*7/16
            pixel[i-1][j+1] += error*3/16
            pixel[i][j+1]   += error*5/16
            pixel[i+1][j+1] += error*1/16


    return output
```

3 - 4 results

Colorspaces

4.1 - R,G,B plots

```
1    indoor_image_path = 'indoor.png'
2    indoor_image_out_path = "indoor_result"
3    indoor_BGR_image = cv2.imread(indoor_image_path)
4
5    indoor_B_channel,
        indoor_G_channel,indoor_R_channel =
        cv2.split(indoor_BGR_image)
6
7    cv2.imwrite(os.path.join(indoor_image_out_path,
8    'red_channel.png'),indoor_R_channel)
9    cv2.imwrite(os.path.join(indoor_image_out_path,
10   'green_channel.png'),indoor_G_channel)
```

```
    cv2.imwrite(os.path.join(indoor_image_out_path,
    'blue_channel.png'),indoor_B_channel)



    outdoor_image_path = 'outdoor.png'
    outdoor_image_out_path = "outdoor_result"
    outdoor_BGR_image = cv2.imread(outdoor_image_path)

    outdoor_B_channel,
       outdoor_G_channel,outdoor_R_channel =
       cv2.split(outdoor_BGR_image)

    cv2.imwrite(os.path.join(outdoor_image_out_path,
    'red_channel.png'),outdoor_R_channel)
    cv2.imwrite(os.path.join(outdoor_image_out_path,
    'green_channel.png'),outdoor_G_channel)
    cv2.imwrite(os.path.join(outdoor_image_out_path,
    'blue_channel.png'),outdoor_B_channel)
```

4.2 - L,A,B plots

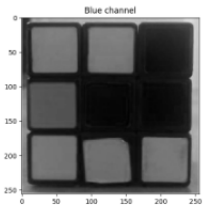
```
    indoor_image_path = 'indoor.png'
    indoor_image_out_path = "indoor_result_LAB"
    indoor_BGR_image = cv2.imread(indoor_image_path)
    indoor_LAB_image =
       cv2.cvtColor(indoor_BGR_image,cv2.COLOR_BGR2LAB)


    cv2.imwrite(os.path.join(indoor_image_out_path,
    'LAB_colormap.png'),indoor_LAB_image)
    # cv2.imshow("LAB colormap",indoor_LAB_image)
    # cv2.waitKey(0)

    L_channel, G2R_channel,B2Y_channel =
       cv2.split(indoor_LAB_image)

    cv2.imwrite(os.path.join(indoor_image_out_path,
    'L_channel.png'),L_channel)
    cv2.imwrite(os.path.join(indoor_image_out_path,
    'G2R_channel.png'),G2R_channel)
    cv2.imwrite(os.path.join(indoor_image_out_path,
    'B2Y_channel.png'),B2Y_channel)


```

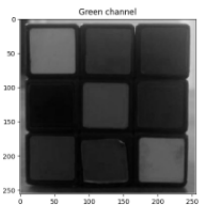
```
22
23      outdoor_image_path = 'outdoor.png'
24      outdoor_image_out_path = "outdoor_result_LAB"
25      outdoor_BGR_image = cv2.imread(outdoor_image_path)
26      outdoor_LAB_image =
            cv2.cvtColor(outdoor_BGR_image,cv2.COLOR_BGR2LAB)
27
28      cv2.imwrite(os.path.join(outdoor_image_out_path,
29      'LAB_colormap.png'),outdoor_LAB_image)
30      # cv2.imshow("LAB colormap",indoor_LAB_image)
31      # cv2.waitKey(0)
32
33      L_channel, G2R_channel,B2Y_channel =
            cv2.split(outdoor_LAB_image)
34
35      cv2.imwrite(os.path.join(outdoor_image_out_path,
36      'L_channel.png'),L_channel)
37      cv2.imwrite(os.path.join(outdoor_image_out_path,
38      'G2R_channel.png'),G2R_channel)
39      cv2.imwrite(os.path.join(outdoor_image_out_path,
40      'B2Y_channel.png'),B2Y_channel)
```

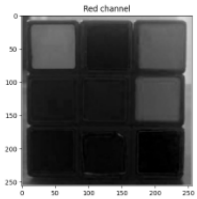
4.3 - RGB vs LAB

RGB:

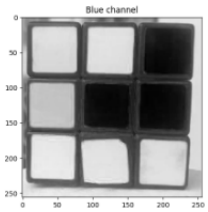
- indoor:

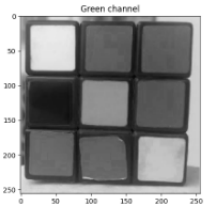


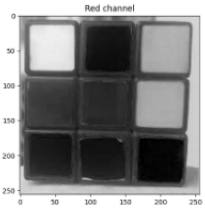




- Outdoor:

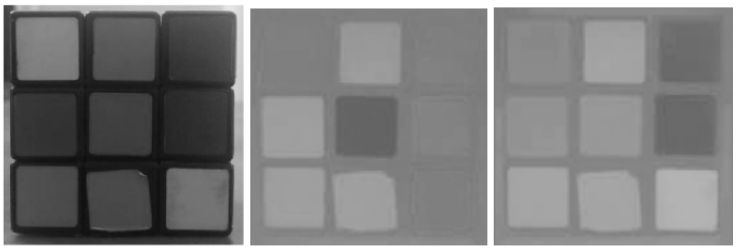




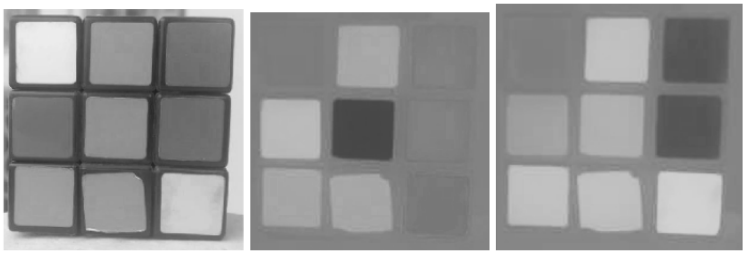


LAB:

- Indoor:
  - L_channel G2R_channel B2Y_channel



- Outdoor:
  - L_channel G2R_channel B2Y_channel



Q: Which color space (RGB vs. LAB) better separates the illuminance (i.e., total amount of light) change from other factors such as hue?

A: LAB

Q: Why?

A: LAB has one channel which used to express the brightness specifically, this channel will not be influenced by the color of the object and separate luminance from color information. So when we use LAB, we can irectly compare the L channel. And the B and A channels are almost the same even in the different illumination conditions, since they only repress the color.

4.4 - Two images and their Luminance plots

Bright: RGB L_channel

Dark: RGB L_channel